12.

```
#define ARRAY_SIZE 24*1024 /* i.e., 24KB */
#define NUM_LOOPS 10000000
/*****************************************************************************
 *  I got 100% miss rate as a 4-way associative and a 0% miss rate as a 2-way associative
 *  Notice that the array is an array of characters.  This means that
 *  each item in the cache is exactly 1 byte.  This makes it easy to
 *  identify data items that will or will not conflict in the cache.
 *  For example, in an 8KB direct-mapped cache, array bytes 0 and 8192
 *  will conflict.
 *
 *  This program simply iterates through each byte in the cache 100
 *  times.
 *
 *****************************************************************************/

int main()
{
  _Alignas(64)  /* make sure that the array aligns with the cache. */
  char array[ARRAY_SIZE];
  register int outer_loop;
  register int solution = 0;
  for (outer_loop = 0; outer_loop < NUM_LOOPS; outer_loop++)
    {
            solution *= array[0] + array[8192] + array[16384];
    }
  return solution;
}
```

13.

```
#define ARRAY_SIZE 16*1024 /* i.e., 16KB */
#define NUM_LOOPS 10000000
/*****************************************************************************
 *
 *  Notice that the array is an array of characters.  This means that
 *  each item in the cache is exactly 1 byte.  This makes it easy to
 *  identify data items that will or will not conflict in the cache.
 *  For example, in an 8KB direct-mapped cache, array bytes 0 and 8192
 *  will conflict.
 *
 *  This program simply iterates through each byte in the cache 100
 *  times.
 *
 *****************************************************************************/

int main()
{
  _Alignas(64)  /* make sure that the array aligns with the cache. */
```

```
  char array[ARRAY_SIZE];
  register int outer_loop;
  register int solution = 0;
  for (outer_loop = 0; outer_loop < NUM_LOOPS; outer_loop++)
    {
            solution *= array[0] + array[8192];
    }
  return solution;
}
```

14.

15.